

The Isabelle Prover IDE after 10 years of development

Makarius Wenzel
<https://sketis.net>

August 2018



History of Prover Interaction

TTY loop (\approx 1979)

```
Terminal
File Edit View Terminal Tabs Help
Welcome to Isabelle/HOL (Isabelle2013: February 2013)
> theory A imports Main begin
theory A
> lemma "x = x";
proof (prove): step 0

goal (1 subgoal):
  1. x = x
> █

Terminal
File Edit View Terminal Tabs Help
Welcome to Coq 8.4pl2 (September 2013)

Coq < Lemma test: forall (A: Type) (x: A), x = x .
1 subgoal

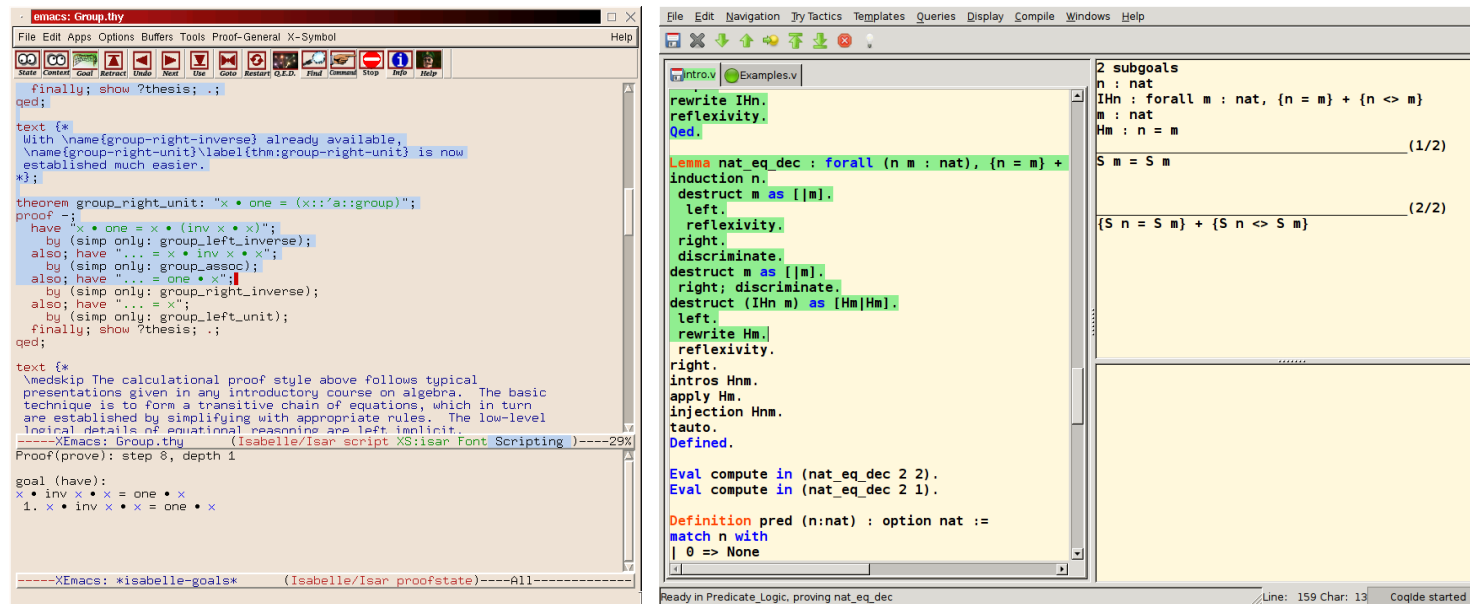
=====
  forall (A : Type) (x : A), x = x
test < █
```



(Wikipedia: K. Thompson and D. Ritchie at PDP-11)

- user drives prover, via **manual copy-paste**
- **synchronous and sequential**

Proof General and clones (\approx 1999)



- user drives prover, via automated copy-paste and undo
- synchronous and sequential

CoqIDE (\approx 2016)

The screenshot shows the CoqIDE interface with a menu bar (File, Edit, View, Navigation, Try Tactics, Templates, Queries, Tools, Compile, Windows, Help) and a toolbar. The main editor displays a Coq script for proving properties of natural numbers. The script includes lemmas for comparison (lt, le), antisymmetry, and maximum. The right-hand pane shows the current goals, which are subgoals of the proof. The bottom status bar indicates the current line and character position, and the Coq engine's status.

```
File Edit View Navigation Try Tactics Templates Queries Tools Compile Windows Help

Arith.v Arith_base.v PeanoNat.v

revert m; induction n; destruct m; simpl; rewrite rlm; split; auto; easy.
Qed.

Lemma compare_lt_iff n m : (n ?= m) = Lt <-> n < m.
Proof.
  revert m; induction n; destruct m; simpl; rewrite ?IHn; split; try easy.
  - intros _; apply Peano.le_n_S, Peano.le_0_n.
  - apply Peano.le_n_S.
  - apply Peano.le_S_n.
Qed.

Lemma compare_le_iff n m : (n ?= m) <-> Gt <-> n <= m.
Proof.
  revert m; induction n; destruct m; simpl; rewrite ?IHn.
  - now split.
  - split; intros; apply Peano.le_0_n. easy.
  - split; now destruct l; inversion l.
  - split; intros; now apply Peano.le_n_S. now apply Peano.le_S_n.
Qed.

Lemma compare_antisym n m : (m ?= n) = CompOpp (n ?= m).
Proof.
  revert m; induction n; destruct m; simpl; trivial.
Qed.

Lemma compare_succ n m : (S n ?= S m) = (n ?= m).
Proof.
  reflexivity.
Qed.

(* BUG: Ajout d'un cas * après preuve finie (deuxième niveau +++*) :
 * ---> Anomaly: Uncaught exception Proofview.IndexOutOfRange(). Please report. *)

(** ** Minimum, maximum **)

Lemma max_l : forall n m, m <= n -> max n m = n.
Proof.
  exact Peano.max_l.
Qed.

Lemma max_r : forall n m, n <= m -> max n m = m.
Proof.
  exact Peano.max_r.
Qed.
```

2 subgoals
n : nat
IHn : forall m : nat, (n ?= m) <-> Gt <-> n <= m
m : nat
H : n <= m
S n <= S m (1/2)
n <= m (2/2)

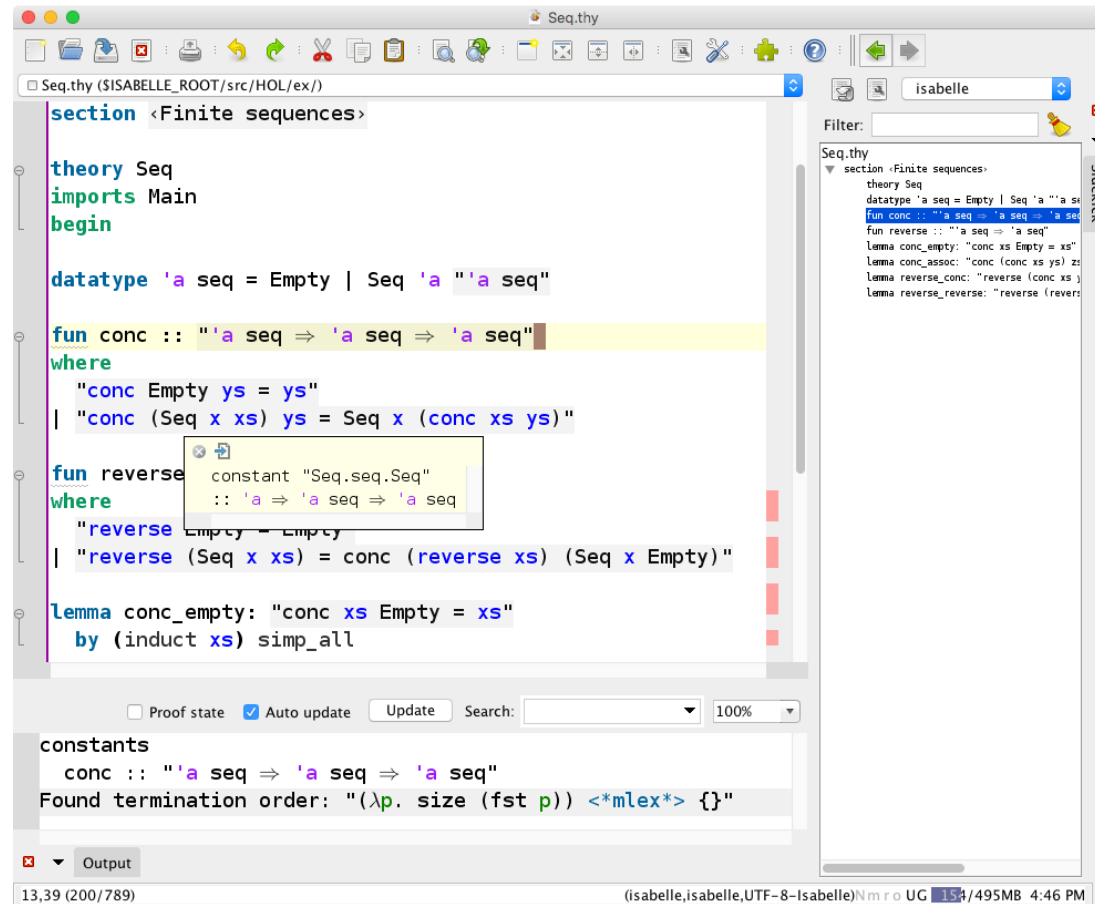
Messages Errors Jobs

Ready in Nat, proving compare_le_iff Line: 211 Char: 18 Coq is ready 0 / 0

- more formal interaction protocol
- recent support for asynchronous proofs

Isabelle/PIDE/jEdit 10.0 (August 2018)

- stateless document model
- asynchronous interaction
- continuous checking
- parallel processing
- scalable applications



Isabelle/PIDE timeline

Parallel Isabelle

- 2005 “free lunch is over”: [multicore CPUs](#) become mainstream
- 2006–2008 Isabelle + Poly/ML support parallel threads

Isabelle/jEdit

- 2008–2010: experimental versions of Isabelle/jEdit Prover IDE
- October 2011: release of [Isabelle/jEdit 1.0](#)
- October 2014: [discontinued](#) Isabelle TTY and Proof General
- August 2018: [Isabelle/jEdit 10.0](#) as “filthy-rich client”

Isabelle/VSCode

- October 2017: release of [Isabelle/VSCode 1.0](#)
- August 2018: release of [Isabelle/VSCode 1.1](#)

Prover IDE architecture

PIDE principles (2008)

Approach:

Prover supports asynchronous document model natively

Editor continuously sends source edits and receives markup reports

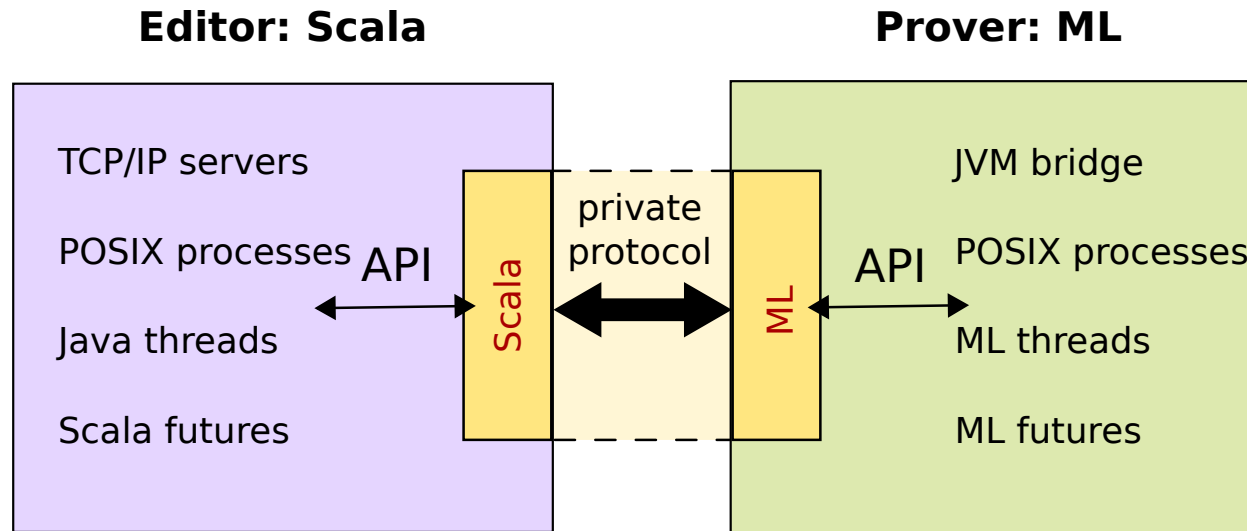
Tools may participate in document processing and markup

User composes document — assisted by rendering of PIDE markup

Challenge: introducing *genuine interaction* into ITP

- many *conceptual* problems
- many *technical* problems
- many *social* problems

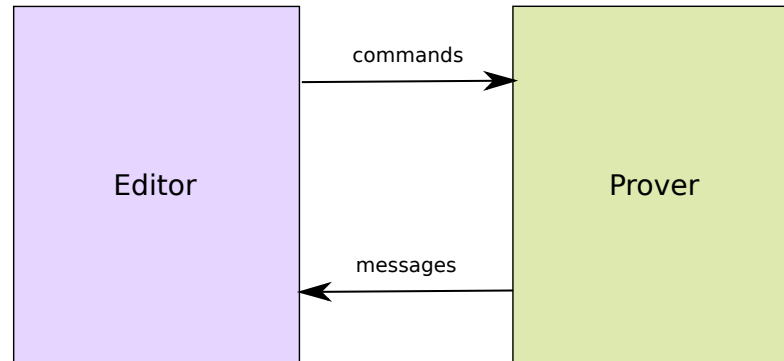
The connectivity problem



Design principles:

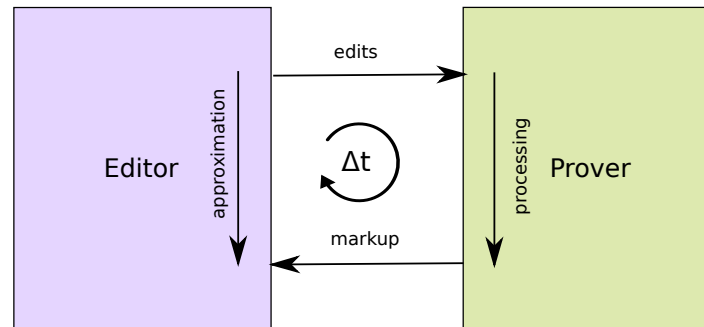
- **private** protocol for prover connectivity
(asynchronous interaction, parallel evaluation)
- **public** Scala API
(timeless, stateless, static typing)

PIDE protocol functions



- *type protocol_command = name → input → unit*
 - *type protocol_message = name → output → unit*
 - **outermost state** of protocol handlers on each side (pure values)
 - **asynchronous streaming** in each direction
- editor and prover as **stream-procession functions**

Approximative rendering of document snapshots

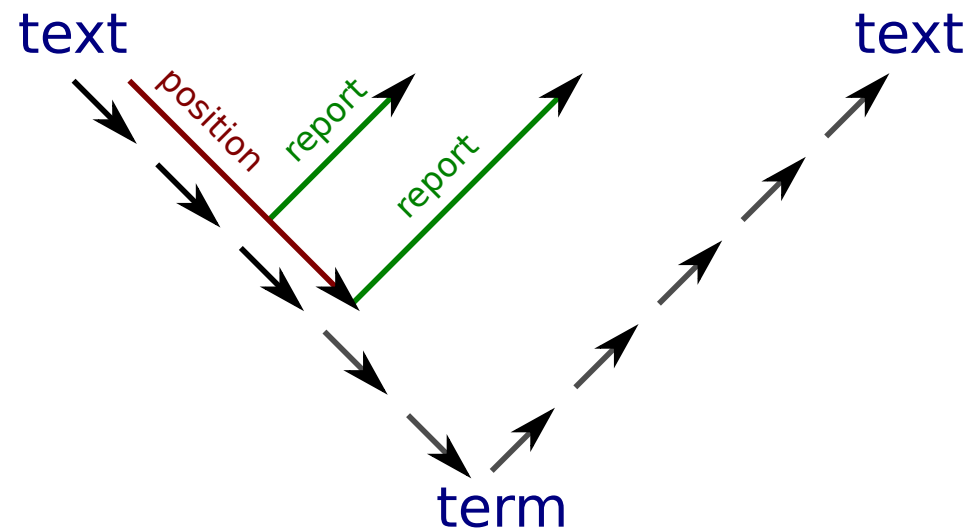


1. editor knows text T , markup M , and edits ΔT (produced by user)
2. apply edits: $T' = T + \Delta T$ (**immediately** in editor)
3. formal processing of T' : ΔM after time Δt (**eventually** in prover)
4. temporary approximation (**immediately** in editor):
 $\tilde{M} = \text{revert } \Delta T; \text{retrieve } M; \text{convert } \Delta T$
5. convergence after time Δt (**eventually** in editor):
 $M' = M + \Delta M$

Markup reports

Problem: round-trip through several sophisticated syntax layers

Solution: execution trace with **markup reports**



PIDE application: Isabelle/jEdit

Building blocks

jEdit: <http://www.jedit.org>

- sophisticated [text editor](#) implemented in Java

Scala/JVM: <https://www.scala-lang.org>

- higher-order functional-object-oriented programming

Isabelle/Scala/PIDE:

- general framework for Prover IDEs
- with [parallel and asynchronous](#) document processing

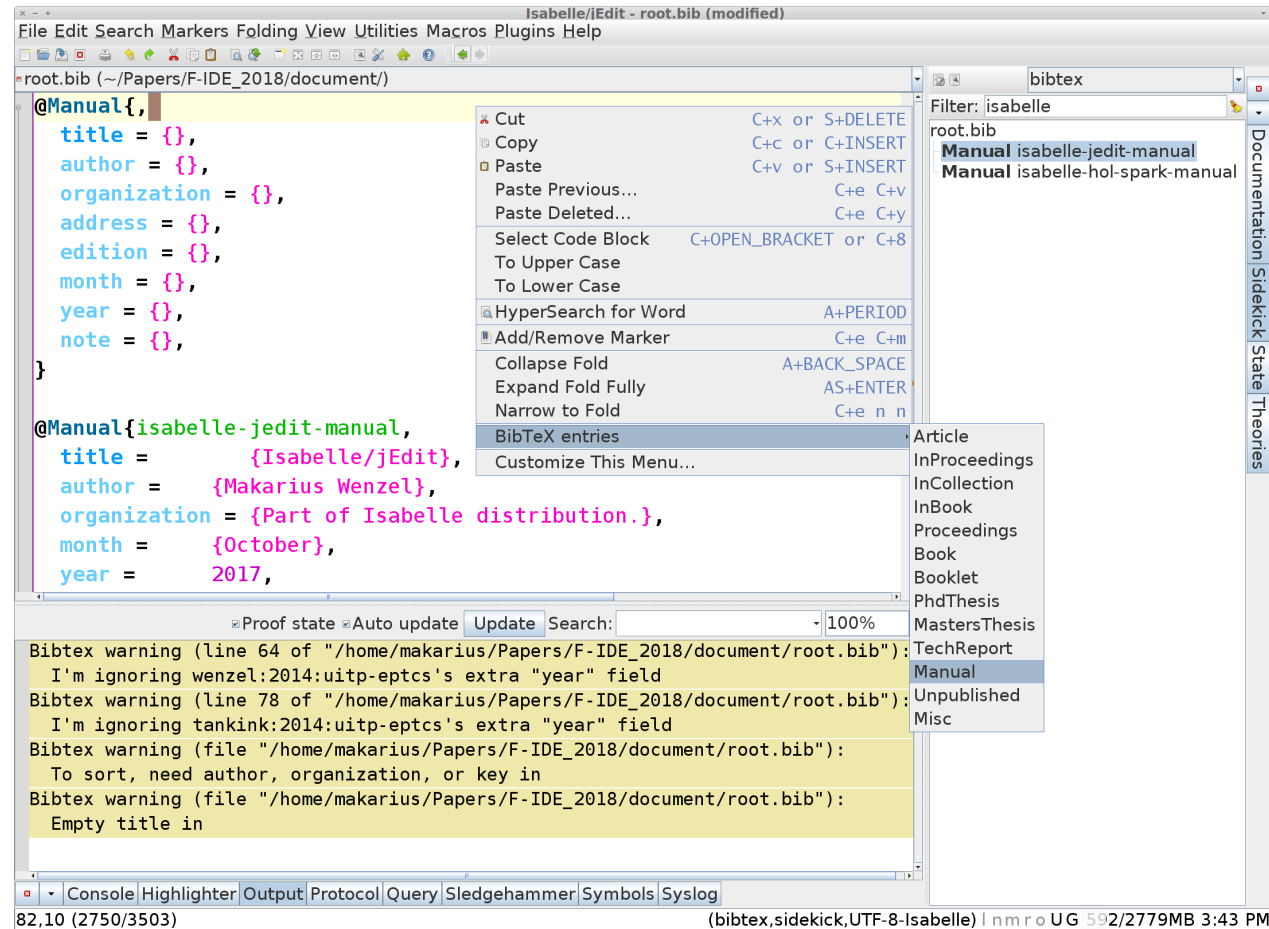
Isabelle/jEdit:

- [filthy rich client](#): requires 4–8 GB memory, 2–4 CPU cores
- main example application of the PIDE framework
- default user-interface for Isabelle

Notable features of Isabelle/jEdit

- good text rendering, with Isabelle fonts (symbols from $\text{T}_{\text{E}}\text{X}$)
- smooth input methods for non-ASCII symbols
- text indentation and folding
- various tree views: outline, context, markup
- panels for Output, State, Query etc.
- nested tooltips and hyperlinks
- highlighting of formal scopes (“def” vs. “ref” positions)
- completion for syntax (editor) and semantics (prover)
- add-on tools: Quickcheck, Sledgehammer etc.
- document file dependencies, including external **ML_file**, **SML_file**
- Isabelle/ML IDE with source-level debugger
- PIDE self-application to Isabelle/ML/Pure bootstrap
- conventional document structure with semantic IDE for Bib $\text{T}_{\text{E}}\text{X}$

Example: Semantic IDE for BibTeX

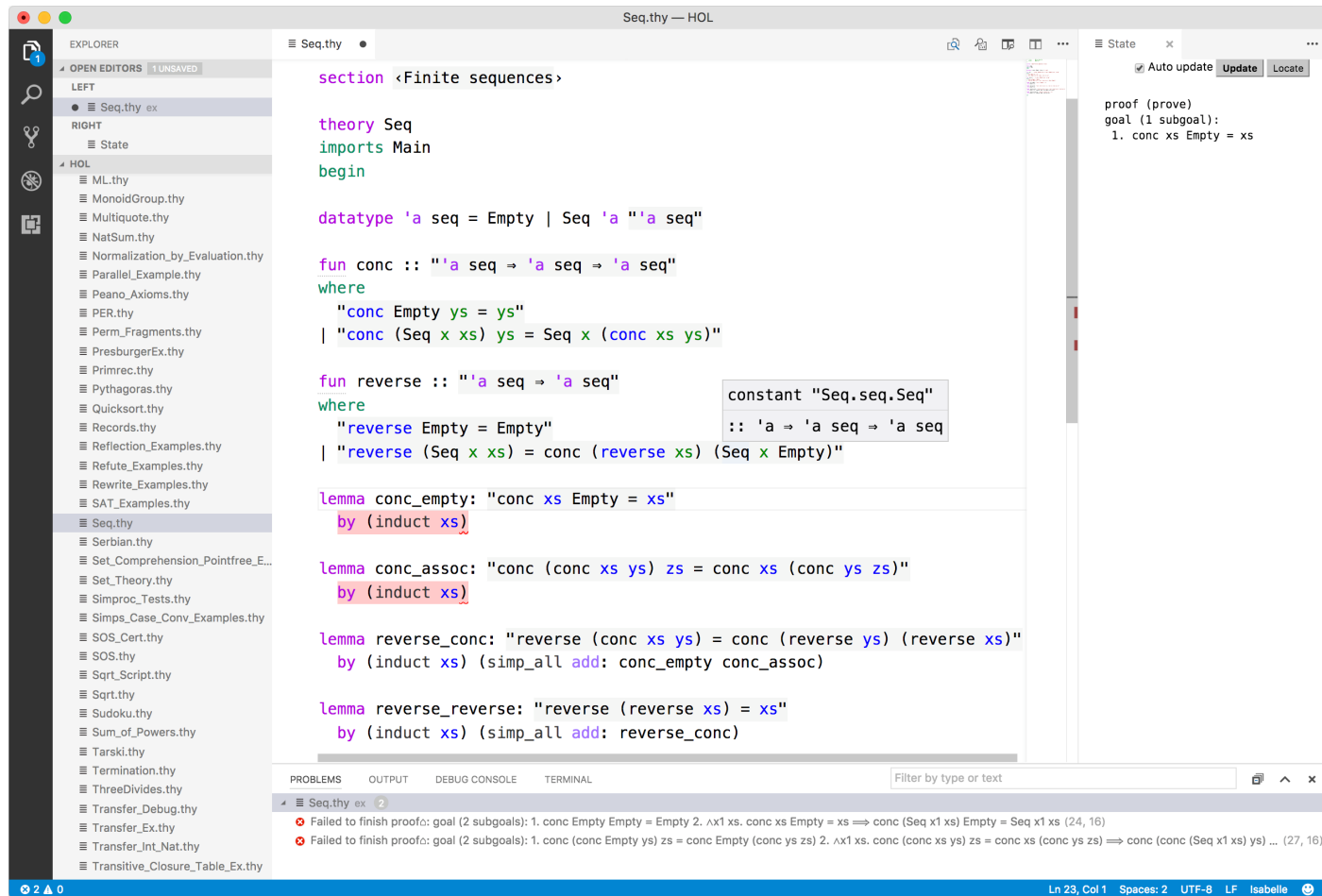


PIDE application: Isabelle/VSCode

Building blocks

- VSCode editor platform:
 - recent open-source project by Microsoft
 - “Code editing. Redefined. Free. Open Source. Runs everywhere.”
 - based on [Electron](#) application framework
 - with [Node.js](#), [Chromium](#) browser, [V8](#) JavaScript engine
 - IDE for [TypeScript](#) in TypeScript (typed JavaScript)
- Isabelle/Scala/PIDE:
 - slightly reworked for multiple front-ends
 - Language Server Protocol based on JSON-RPC
- VSCode Isabelle extension: via VSCode marketplace

Isabelle/VSCode 1.0 (October 2017)



Notable features of Isabelle/VSCode

- static syntax tables for Isabelle .thy and .ML files
- implicit dependency management and formal checking of sources
- text overview lane with formal status
- prover messages within the source text (errors, warnings etc.)
- semantic text decorations: colors for free/bound variables, inferred types etc. (Language Server Protocol extension)
- highlighting of formal scopes (“def” vs. “ref” positions)
- proof state output via VSCode message channel or GUI panel
- HTML preview via separate GUI panel
- completion for syntax (editor) and semantics (prover)
- spell-checking of informal texts

Isabelle/jEdit 10.0 vs. Isabelle/VSCode 1.1

Isabelle/jEdit: “game engine”

- scalable application
- Java with Swing GUI
- multiple threads
- simple text buffer model
- free-form layered painting (Graphics2D)

Isabelle/VSCode: “smart text editor”

- minimal experiment
- JavaScript with HTML/CSS
- cooperative multitasking
- rich text buffer model
- restricted text decoration model (CSS)

PIDE application: Isabelle server

Isabelle server

Approach:

- Isabelle/Scala as “terminate stay-resident” application
 - socket communication with JSON or YXML protocol
 - **multiple servers** per user (named database entries)
 - **multiple sessions** per server (ML processes)
 - **multiple use_theories** invocations per session (PIDE edits)
- <https://sketis.net/2018/the-isabelle-server-responsive-control-of-prover-sessions>

Look-and-feel:

- **file-system** state turned into PIDE document updates
- asynchronous **command-loop** with explicit task identification
- no GUI

Isabelle server 1.0 (August 2018)

```
$ isabelle server &  
$ isabelle client  
help  
session_start {"session": "HOL"}  
use_theories {"session_id": ..., "theories": ["~/src/HOL/ex/Seq"]}  
session_stop {"session_id": ...}  
shutdown
```

Note:

- manual experimentation: need to provide commands slowly
- program control: need to handle asynchronous notifications

Conclusions

Future work

Isabelle/PIDE continued:

- Isabelle/jEdit: more [scaling](#), e.g. all of AFP in one PIDE session
- Isabelle/VSCoDe: better [integration](#) as standalone application
- Isabelle server: [SSH](#) tunneling and PIDE as “cloud” service

Further scaling:

- scaling up — big Isabelle/ML/Scala/Java/PIDE session on server
- scaling down — small PIDE front-end on client, e.g. via Scala.js

Open problem:

- Missing PIDE support for Coq, HOL4, HOL Light, . . .
- I am still open for collaborations!